(2)

A Final Report
Contract No. N00014-89-J1699

A HIGH PERFORMANCE COMPUTER ARCHITECTURE FOR EMBEDDED AND/OR MULTI-COMPUTER APPLICATIONS

Submitted to:

Scientific Officer Code: 1133
Andre M. Van Tilborg
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

Submitted by:

William A. Wulf
Professor

Anita K. Jones
Professor and Chair

Report No. UVA/525435/CS91/101
September 1990

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF
ENGINEERING
& APPLIED SCIENCE

University of Virginia
Thornton Hall
Charlottesville, VA 22903

UNIVERSITY OF VIRGINIA
School of Engineering and Applied Science

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 600. There are 160 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Applied Mechanics, Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 17,000), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Physics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.

A Final Report
Contract No. N00014-89-J1699

A HIGH PERFORMANCE COMPUTER ARCHITECTURE FOR EMBEDDED AND/OR MULTI-COMPUTER APPLICATIONS

Submitted to:

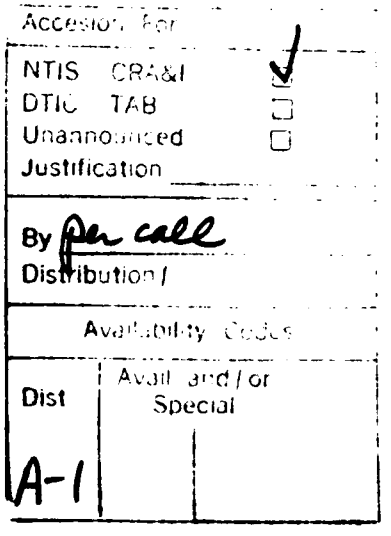
Scientific Officer Code: 1133
Andre M. Van Tilborg
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

Submitted by:

William A. Wulf
Professor

Anita K. Jones
Professor and Chair

Department of Computer Science
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA



STATEMENT "A" per Dr. Andre Van Tilborg
ONR/Code 1133
TELECON 9/13/90 VG

Report No. UVA/525435/CS91/101
September 1990

Copy No.____

Final Performance Report
DARPA/ONR N00014-89-J1699
A High Performance Computer Architecture for Embedded and/or Multi-Computer Applications

Wm. A. Wulf & Anita K. Jones
Department of Computer Science
School of Engineering and Applied Science
University of Virginia

June 1990

Under this grant we have produced a set of studies, simulations, software, performance data, documents and designs. They are the result of refining the architectural definition of the WM machine, implementing a portion of it, and analyzing various aspects of it when realized in a high performance implementation.

1. Architectural Refinement

The WM architecture, as extant at the beginning of the project, was evaluated and modified in a number of respects. Two extensive manuals were written to capture the final design -- one of which is in MILSTD form. Analyses of the performance of the architecture were performed as part of the evolving refinement of the architecture.

The analysis of the performance of the architecture was an especially challenging problem. The usual approach is to compare implementations rather than architectures; that is what is done when one runs a set of benchmarks on, say, SPARC and MIPS processors. Such comparisons are interesting for the consumer, but they measure the composite properties of many implementation decisions as well as inherent architectural characteristics. We therefore took two separate approaches:

- evaluation of the use of novel WM features by real programs, and
- development of a methodology for architectural comparison.

The first approach essentially asks how effective features such as streaming are, and in the process determines reasonable values for such implementation-determined parameters such as FIFO depth. The second tries to get directly at inherent architectural characteristics by assuming the same implementation techniques are used (even if costly to do so) on all architectures.

Both approaches will be discussed in forthcoming reports.

Four kinds of modifications were made in the process of refining the architecture:

- minor changes for consistency or implementability; these will not be discussed further
- parameterization of the definition to permit a broader class of implementations than one normally encounters, thus spanning a broader class of embedded applications,
- addition of vector processing capability, and
- addition of a novel and flexible/efficient set of facilities for reducing operating system overhead.

Each of these (except the first) will be discussed in more detail below.

(1) Parameterization: An instance of the WM architecture is characterized by a three-tuple, <i,f,v>, where

i	is the "width" (in bits) of the integer arithmetic unit,
f	is the "width" (in bits) of the floating point arithmetic unit, and
v	is the "width" (in bits) of the vector processing unit

Thus, for example, $WM_{<16,32,0>}$ is a computer with 16-bit integer arithmetic, 32-bit floating point arithmetic, and no vector unit. Data types wider than the specified arithmetic unit are not supported (e.g., 32-bit integers are not supported on $WM_{<16,32,0>}$), thus there is no extra cost incurred for implementing smaller versions of the architecture. Thus, for example, $WM_{<16,0,0>}$ is a true 16-bit microcomputer such as one might want in certain MCCR applications.

(2) Vector Processing: A third arithmetic unit was added to the architecture that may operate asynchronously with the Integer and Floating execution units. The definition is parameterized by the unit's arithmetic width, as discussed above, but also by the "block size" (the number of vector elements per vector register). The vector operations were carefully defined to be neutral to implementation strategies (eg, the number of pipes used to implement them).

An innovative combination of the vector processing paradigm with WM's "streaming" semantics achieves the effect of "strip mining" at run time (usually a compiler optimization).

(3) Operating System Facilities: The operating system can be, and too often is, a bottleneck in high performance computing situations. The WM design philosophy has been to permit moving as much functionality as possible safely into user-level code -- thus avoiding the overhead of OS entry/exit as well as avoiding circumlocutions due to mismatches between what the application needs and the system provides.

The keys to the design are: (a) the total elimination of the notion of "modes" (eg, user vs system mode), and (b) the substitution of typed and protected memory objects with corollary operations. A special report on this aspect of the design is in preparation.

2. Hardware Implementation Efforts

Our hardware implementation efforts focused on three primary research topics: (1) the implementation of the Stream Control Unit (SCU), (2) the preliminary investigation

of the WM memory system, and (3) the modeling of the WM architecture using the VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL). In each case, we achieved the objectives and completed the deliverables outlined in the proposal.

We selected the SCU for implementation because it is a unique feature of the architecture. In addition, the SCU was reasonable to design and fabricate using the available MOSIS (Metal Oxide Semiconductor Integration Service) MOS (Metal Oxide Semiconductor) technology. We consider the memory system to be an important aspect of the WM computer because of the high memory bandwidth needed to support the architecture's extensive concurrency. Our investigations during this research project were constrained to those components, such as caches, which provide rapid responses to processor memory requests rather than main memory systems which are accessed through an arbitrated system bus. Finally, the VHDL model was created as the first step in the implementation process. The model describes the WM Instruction Set Architecture (ISA) in a form which can be simulated to determine performance and functional information. Each of the three major hardware efforts is described in one of the following three sections.

2.1. Implementation of the Stream Control Unit

The WM computer architecture defines a method of performing fast memory accesses called streaming. The hardware unit which implements streaming is referred to as the Stream Control Unit (SCU). The primary function of the SCU is the fast generation of addresses for Integer Execution Unit (IEU) and Floating Execution Unit (FEU) operations. The goals of designing the SCU were: (1) to develop a set of reusable elements that could be employed to implement other features of the WM computer, (2) to obtain realistic estimates of speed, power, area, and pin counts for the SCU and other modules, and (3) to evaluate multiple SCU designs and determine an optimal implementation. Each goal has been achieved.

The present implementation of the SCU consists of three functional units: (1) the control logic, (2) a multiplexer network, and (3) the address generation circuitry. The control logic consists of an arbitration network, an internal controller, and an external controller. The external controller is responsible for providing an interface between the SCU and the IEU, while the internal controller dictates the sequencing of all operations within the SCU. The arbitration unit determines which of the eight address generation units gains access to memory. The multiplexor network provides the physical mechanism for selecting one of the eight addresses to provide to the address bus. Finally, the SCU contains eight address generation units (one for each input and output data FIFO).

Five unique integrated circuits (ICs) were designed to create the prototype implementation of the SCU. The control logic was partitioned among three ICs, and the remaining logic was divided among two ICs. Based on the IC designs which were developed, a complete 32-bit SCU is estimated to require approximately 50,000 transistors using the present SCU architecture. Tradeoffs between speed and size can substantially reduce the transistor count, if necessary. We implemented a prototype SCU

with an 8-bit datapath and four address generation units using our ICs. The prototype was thoroughly tested and verified to function correctly. The next step in the research process will be the integration of the SCU into a single IC.

2.2. The WM Memory System

The WM computer architecture provides a number of features which allow great flexibility in the design of high-bandwidth memory subsystems. The investigations we have performed thus far have been limited to those parts of the memory system which provide rapid response to processor memory requests (for example, caches) rather than main memory systems which are accessed through an arbitrated system bus.

As long as certain access orderings are preserved, a WM cache may consist of: (1) separate memory units for each memory master, (2) a single memory unit with an arbitrator to determine the proper access order, or (3) a hybrid of shared and dedicated units. High-performance WM processors, for example, may be designed with several distinct caches so that many memory requests can be serviced simultaneously, while lower-performance implementations may have a single memory unit with an arbitration circuit to coordinate the memory access. An important feature of the WM architecture is the fact that memory address generation often leads the memory access by several instructions. Consequently, the lead time allows the consideration of large, slow caches for servicing at least some of the WM memory masters.

We have studied three primary implementations of a memory subsystem for the WM computer: (1) a two-cache system, (2) a three-cache system, and (3) an n-cache system. Although a WM computer may be designed with only a single cache, it is anticipated that the architecture will be better served with a minimum of one data cache and one instruction cache (the two-cache system). The data cache would service the IEU, FEU, and the SCU, while the instruction cache services the Instruction Fetch Unit (IFU). Because the data coherency requirements between integer data and floating data only need to be guaranteed following explicit instructions from the programmer, a reasonable improvement from the two-cache system is a three-cache system in which separate caches exist for the IFU, IEU, and the FEU. In the three-cache system, SCU accesses would be handled via the cache associated with the appropriate unit (IEU or FEU). Finally, a high-performance memory system would include separate caches for the IFU, IEU load/store operations, FEU load/store operations, and multiple caches for the various stream operations.

The large number of memory masters prescribed by the WM architecture as well as the lead time provided between memory address generation and memory access promises to allow novel cache organizations which may, at times, run counter to traditional approaches. The memory system for a WM computer provides great flexibility to the designer by creating alternatives using various amounts of parallelism in the cache organization. Consequently, the memory system can be tailored to meet the cost and performance needs of the desired WM system.

2.3. VHDL Model of the WM Architecture

A functional model of the WM architecture has been developed using the VHSIC Hardware Description Language (VHDL). The model runs on Sun workstations under the Intermetrics VHDL-1076 toolset and executes WM microcode instructions stored as bit patterns in a Unix text file. Similarly, the state of the model's memory is written to another Unix file at the end of an execution run.

The model is implementation independent in that the WM's architecture is described without including features which would imply or preclude any legal WM implementations. The model serves as a workbench in which the performance effects of varying system buffer sizes, memory delays, operator delays, and so forth, can be observed. In addition, the modular nature of the model will facilitate the gradual transition from implementation-independent to implementation-dependent descriptions as behavioral model elements are replaced with structural descriptions in the model.

The VHDL model has been tested via the application of two programs (the dot product and a complex vector multiplication). The results have shown that the implementation-independent model of the WM computer executes actual WM code and correctly maintains the parallelisms provided by the architecture. By introducing propagation and access delays to selected system modules the model may be used to characterize a particular implementation technology. Therefore, the model permits the designer to simulate the execution of complex programs and extract the desired module utilization information for a particular implementation technology.

3. Compiler/Simulator/Assembler

The WM compiler translates Kernighan and Ritchie C source code into WM assembly language code. The compiler is implemented using a machine independent front-end that generates code for an abstract, stack-based machine. The back-end of the compiler consists of a retargetable optimizer driven by a machine description. The stack-based intermediate language is powerful enough to support additional Algol-like languages such as Pascal and Ada with minor extensions. The back-end can also accommodate these languages and has in fact been used in a commercially available Ada compiler.

The following optimizations are implemented in the optimizer: unreachable code elimination, branch chaining, branch minimization, peephole optimization, common subexpression elimination, constant folding, copy propagation, global register allocation via graph coloring, dead variable elimination, code motion, strength reduction, induction variable elimination, recurrence elimination via software pipelining and instruction scheduling. In addition, WM-specific optimizations including streaming and issuing load and conditional instructions as soon as possible are also implemented. The calling convention uses registers to pass arguments, resulting in a very fast procedure calling mechanism. The code generated by the compiler is of a very high quality--as good as or better than that which can be written by all but the most experienced WM assembly language programmers.

The compiler is written in C and has been ported to over ten different Unix systems. Compilation speed is comparable to many of the highly-optimizing compiler systems currently in use.

The compiler includes a linker which permits source code modules to be compiled independently. A basic C library has been implemented containing a subset of the standard C library. Many of the common I/O and string manipulation routines have been coded and added to the library in order to run the basic test suite used to verify that the compiler produces correct code. Additionally, a basic math library is also available in order to run floating-point oriented benchmarks such as whetstone and linpack.

The WM simulator accepts both memory images and WM assembly code as input. If the user specifies a WM assembly language file (either written by hand or generated by the WM compiler), the simulator first invokes the WM assembler to translate the assembly code to a WM memory image. This memory image can either be executed or dumped out to a file. Currently, the assembler does not support modules. This, however, does not prevent the C compiler from supporting separate module compilation through the use of a separate linker which emits a single WM assembly language file.

The WM simulator supports all of the features needed to develop an operating system. These features include: virtual memory, task control blocks (TCBs), exception handling and protection. The simulator has been tested by executing all of the test programs which comprise the test suite used to test the compiler.

The WM simulator is written in C and uses the curses library to provide a user interface. The simulator is portable and has been tested on five different systems. On a Sun 3/280, the simulator executes WM instructions at a rate of 1000 per second.

Important WM parameters such as the depth of the various FIFOs, the time required to access memory and registers and the probability of cache hits can be easily changed through the user interface. A trace facility is also available to provide information in various levels of detail. These features facilitate experimentation. For example, one can easily execute a set of benchmarks using different memory latencies to get some idea of how tolerant the WM architecture is of slower memory.

4. Operating Systems Achievements

The proposal did not specify milestones involving an operating system. However, as the project moved forward it was determined that a primitive lightweight, multitasking operating system should be explored. The lightweight operating system, *LIGHTOS*, was completed in February 1990. *LIGHTOS* was undertaken for three major reasons. First, *LIGHTOS* was the first WM-specific programming endeavor to use both the WM compiler and simulator. This gave an indication of how both tools interfaced OS kernel code, including operating system-oriented instructions. Second, the underlying structure of *LIGHTOS* is built using the object-oriented paradigm. Thus, components of the scheduler are easily modifiable and replaced with minimum impact. Third, *LIGHTOS* accelerated the learning process for those not previously involved with the WM project and provided hands-on experience of the WM tasking and interrupt handling abilities before attempting a port of a more complex operating system..

LIGHTOS provides a multi-thread execution paradigm within a single shared address space. The user is provided with primitives to initialize and start tasks, as well as the semaphore operations of *initialize*, *wait* and *signal*. *LIGHTOS* uses a timer interrupt handler which provides round-robin service to queued, ready tasks. *LIGHTOS* does not provide I/O support. This is handled by traps to the simulator I/O calls currently supported.

In preparation for future work with real-time applications, three real-time operating systems were studied as candidates for later implementation; VRTX, CHOICES and ARTS. VRTX is a widely used, commercially available, real-time operating system. CHOICES and ARTS are real-time operating systems developed at the University of Illinois and CMU, respectively. Selection of a real-time operating system will be made in the next phase of the project.

BIBLIOGRAPHY

Wulf, Wm. A. The WM Computer Architecture Definition and Rationale, Version 2, University of Virginia Department of Computer Science Technical Report No. TR-88-22, February 22, 1989.

Wulf, Wm. A. The WM Computer Architectures Principles of Operation, University of Virginia Department of Computer Science Technical Report No. TR-90-02, January 1990.

Wulf, Wm. A., and Hitchcock, Charles. The WM Family of Computer Architectures, University of Virginia Department of Computer Science Technical Report No. TR-90-05, March 1990.

Salinas, Maximo H. Implementation-Independent Model of the EM Computer Architecture, Master's Thesis, January 1990.

Pamidi, Sunil. A VLSI Implementation of the Stream Control Unit for the WM Machine, University of Virginia, Department of Electrical Engineering Master of Science Thesis, May 1990.

Grimshaw, Andrew S., et. al. Implementation Independent Architectural Comparison, University of Virginia Department of Computer Science Technical Report TR-90-12, June 20, 1990.

Kester, Peter. Performance of the WM Architecture. University of Virginia Computer Science Technical Report & Masters Thesis (to appear in 1990).

Jones, Anita K., and Wad, Rohit. The WM Computer Architectures: Military Standard Manual, University of Virginia Department of Computer Science Technical Report No. TR-90-19, August 1990.

Davidson, Jack W., and Benitez, Manuel E. Code Generation for Streaming: An Access/Execute Mechanism, University of Virginia Department of Computer Science Technical Report No. TR-90-23, August 2, 1990, submitted to the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April 1991.

Aylor, J. H., Cohoon, J. P., Feldhousen, E. L., and Johnson, B. W. Gate—A Genetic Algorithm for Compacting Randomly Generated Test Sets, invited submission to the International Journal of Computer-Aided Design, July 15, 1990.

Sabat, Shekhar, and Cohoon, J. P. Sharp-Looking Geometric Partitioning, submitted to the European Design Automation Conference, August 1990.